Heuristics

Matthew Guzdial

A*

```
add start to openSet
while openSet is not empty:
    current = openSet.pop()
    if current == goal:
        return reconstruct_path(current)
    closedSet.Add(current)
    for each neighbor of current:
        if neighbor in closedSet:
            continue
        gScore = current.gScore + heuristic(current, neighbor)
        if neighbor not in openSet:
            openSet.add(neighbor)
        else if gScore< openSet.get(neighbor).gScore
            openSet.replace(openSet.get(neighbor), neighbor)
```

Heuristic

- A function that given a state and goal, estimates remaining distance

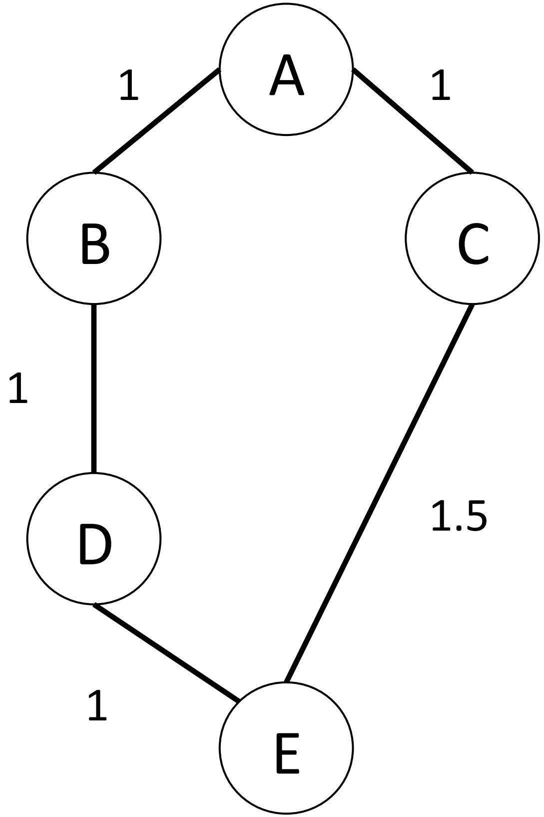
- A* is guaranteed to find the optimal path to the goal *if* the heuristic (*h*) is admissible

What does admissible mean?

Some heuristic (h) is admissible iff for all states n, $h(n) \leq h^*(n)$

- Where h^* is the true distance

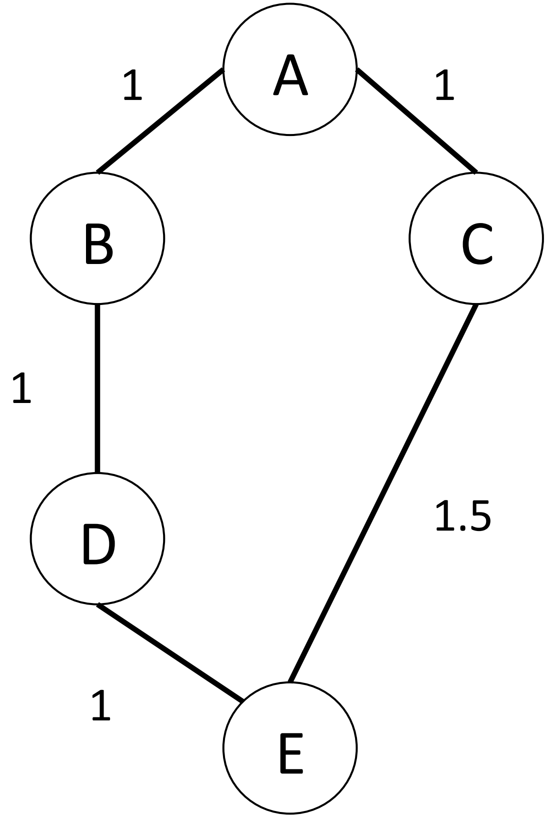
Example: Navigating Roads



- h*(B) = 2
- h*(C) = 1.5

- h(B) = 2
- h(C) = 24

Alternative Heuristic



- h*(B) = 2
- h*(C) = 1.5

If h=Straight line distance

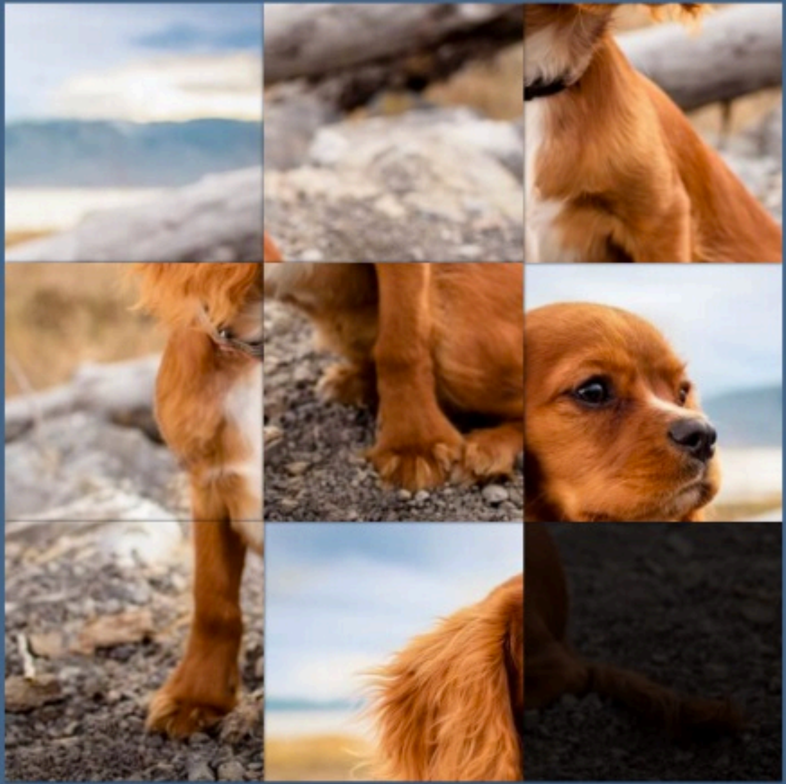
- h(B) = 1.5
- h(C) = 1.5
- Roads can never be shorter than this

You <u>must</u> Prove Admissibility

- If your heuristic is not admissible, we can’t guarantee that A* gives an optimal solution
- Why?

- Can you think of a case where we *wouldn’t* care if our A* solution was non-optimal?

8-Puzzle Game



Show numbers

Shuffle

Solve

Example 2: 8 Puzzle

7	2	4
5		6
8	3	1

	1	2
3	4	5
6	7	8

What would be a good heuristic?

Options

1. #1 tiles out of place
   - Need at least this many moves
   - Best case: each tile is exactly 1 out of place
   - …But probably need many other moves
2. Manhattan Distance
   - # of square blocks you have to walk on a grid
   - (Same justification as above)

How do we choose between heuristics?

Given last slide's two heuristics h_1 and h_2

For initial state s_1

$h_1(s_1) = 8$

$h_2(s_1) = 18$

$h^*(s_1) = 26$

h_2 better than h_1

Why?

h_2 is closer to h^* but for all states n, $h_2(n) \leq h^*(n)$

Informedness

- Degree to which a heuristic *h* captures insight
- Informedness = size of total search space/ avg. # of states explored with heuristic *h*

- Do we want high or low informedness?

Dominance

h_a dominates h_b when $h_b(n) \leq h_a(n) \leq h^*(n)$

- h_b underestimates more
- h_a and h_b are both admissible

Can you think of a heuristic that is always admissible?

- Yes h(n) = 0 for all states n
- But not *informed*
- Admissibility does not guarantee a heuristic is any good (only optimality of solution)
- A non-admissible heuristic can be more informed (but not dominate or admissible), which will give better average time performance

Consistency

- AKA monotonocity: Always increasing/ decreasing
- Admissibility: never overestimate
- Overestimates cause shortcuts

What happens if we underestimate too much?

- Still admissible
- May be fooled into spending too much time in useless part of space
- Will eventually find optimal solution though
- $h(n) = 0$ is just breadth first search

Okay cool, but how do I come up with a heuristic for a problem?

1. In cases where runtime is a concern, should be O(Polynomial) or better

- Relax the problem
  - At any given state, make a simpler problem
  - Solve simpler problem exactly
  - Return true cost of simpler problem

Example: 8 Puzzle

What are the constraints?

Tile can move from square A to square B if:

1. A adjacent to B
2. B is blank

7	2	4
5		6
8	3	1

What if h==$h*$?

Is is still admissible?

- Yes of course!

So we should always strive for that right?

- Nope, if *h* is too expensive

Heuristics Review

- Admissibility: all states n, $h(n) \leq h^*(n)$
  - Must be true to guarantee optimal solution
- Informedness: measure of how little we need to explore (total # of states/# of states we explore)
- Dominance: h_2 dominates h_1, $h_1(n) \leq h_2(n) \leq h^*(n)$
- Consistency: Always increasing or decreasing relative to neighbors

Search

Randomized Optimization

- Intuition: What if we have a heuristic but no goal, how do we pick the best final state?
- Each state is now some representation
  - Note: I say “representations” here, but we can think about literally any data structure here
  - Almost any knowledge representation can be searched over, given some definition of “best”.

Naïve Approach: Generate and Test

Create representation at random and then see how well it performs.

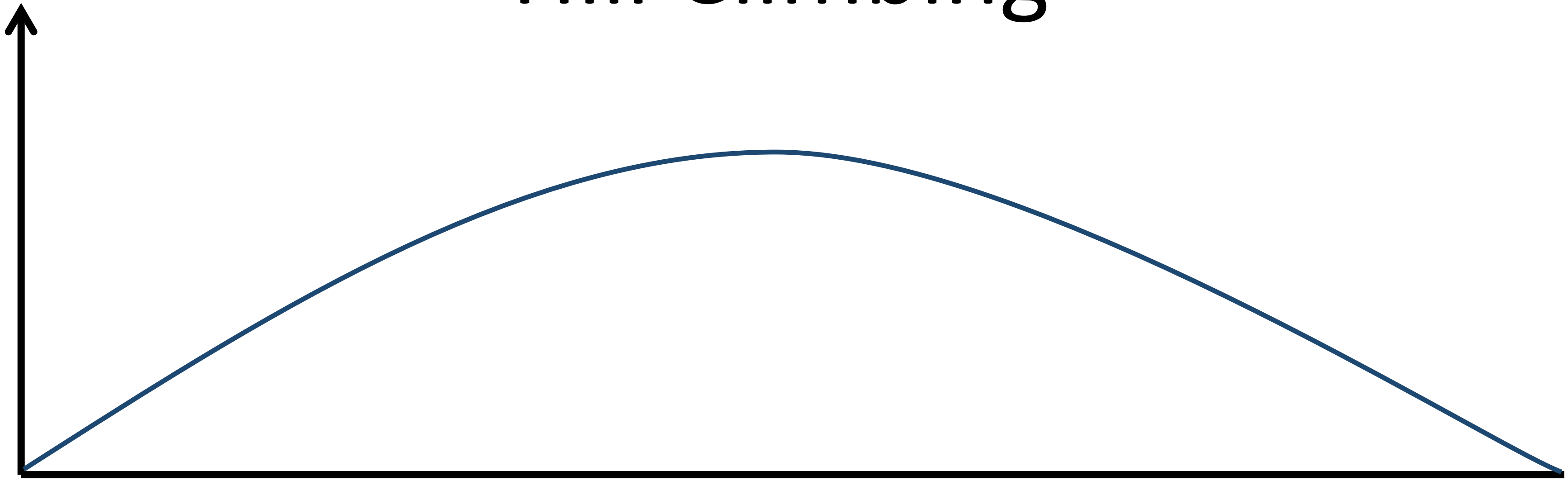
Issues:

- Large hypothesis spaces
- Doesn’t learn anything

Randomized Optimization Approaches

- Hill-climbing aka Greedy Search
- Simulated Annealing
- Genetic Algorithms

Hill Climbing



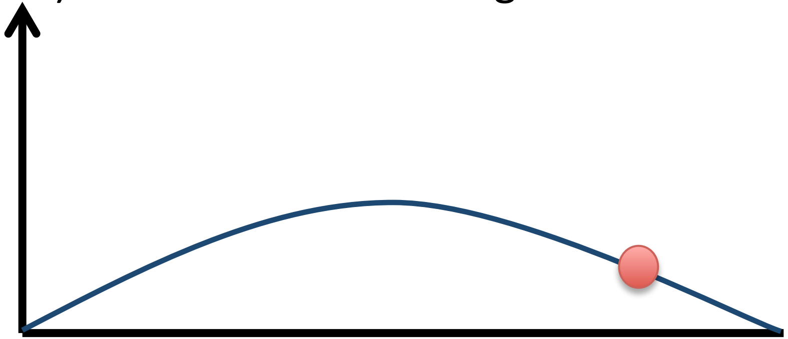
If y-axis is some heuristic and x-axis are representations, we can imagine a pretty simple algorithm to find best representation

Hill Climbing Pseudocode

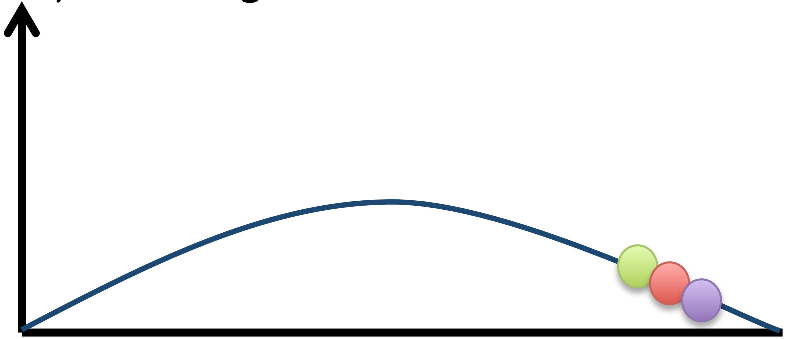
```
curr = random_selection_from_space()
score = heuristic(curr)
prevScore = 0
while score>prevScore:
      prevScore = score
      for n in neighbors(curr):
            if heuristic(curr) < heuristic (n):
                  curr = n
                  score = heuristic(curr)
return curr
```

Example

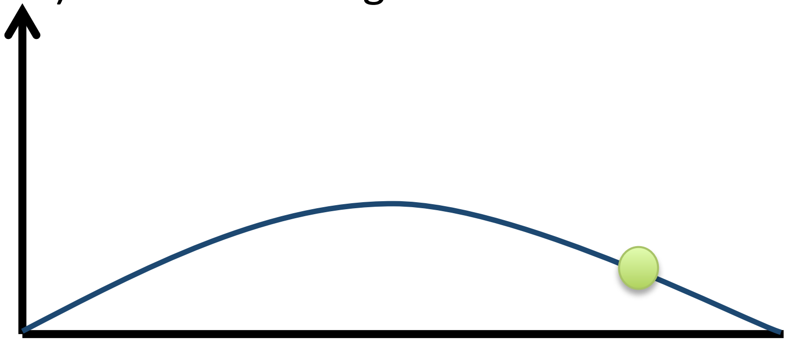
1.) Pick a random starting location



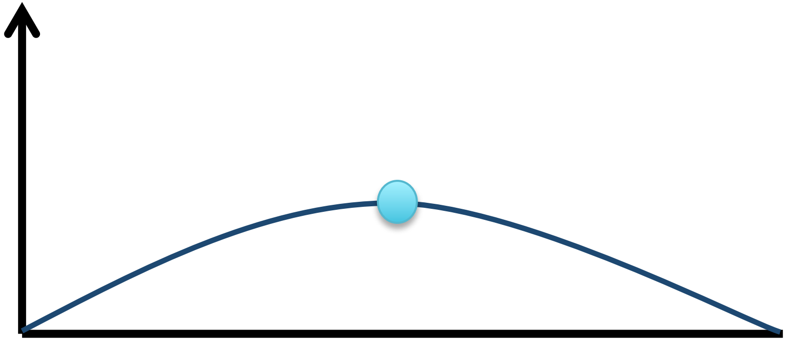
2.) Find Neighbors



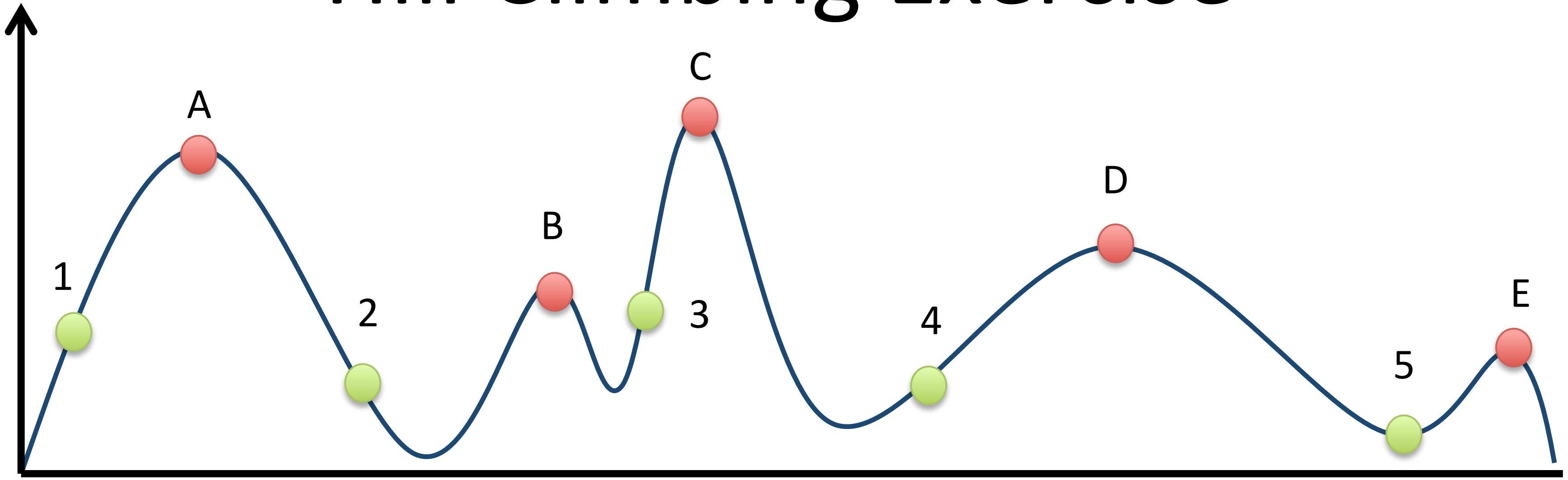
3.) Select best neighbor or finish



Repeat 2 and 3 till….



Hill Climbing Exercise



Given input points 1-5, what points A-E would they end up at following the hill climbing algorithm?

Answer

1. A
2. A
3. C
4. D
5. ??? (depends on neighbor function)
   – e.g. Left or right first? one step or multiple?

Local vs. Global Maxima

- Local Maximum: A point with only worse neighbors
- Global Maximum: A point better than all other points
  - What we really want

Issues

- Choice of neighbor function
  - Deciding between ties
  - What neighbors do we consider
- Local vs. Global Maxima
  - Can we ensure we get a global maxima?
  - What are ways we can improve our chances?

Choice of Neighbor Function

- It’s easy to “see” where to go next if we visualize the entire space.
- But functionally we do not have the entire space already, we generate each neighbor as needed
- Show example on board if they don’t get it.

Neighbor Function Exercise

Representation: 6 bit strings (ex. 101010)

Heuristic: Number of shared bits with 101010

Neighbor Function 1: Flip pairs of adjacent bits

    100000-> 111000 or 010000 but not 101010

Neighbor Function 2: Flip any two bits

    100000-> 111000 or 010000 or 101010

Q: What’s the maxima we’ll reach with the random starting points 010101, 000100, and 110000?

Answers

Q: What's the maxima we'll reach with the random starting points 010101, 000100, and 110000?

N1: 101010 (6), 000010 (4), and 101000 (5)

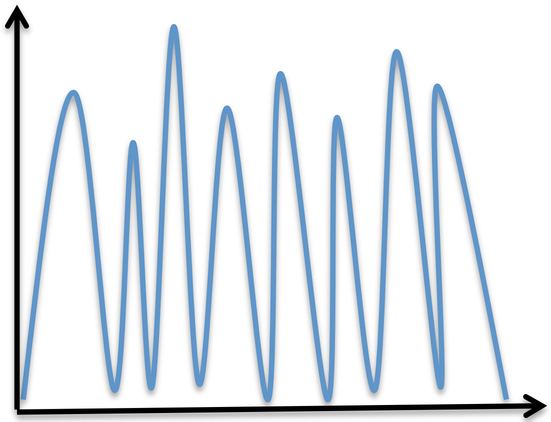
N2: 101010 (6), 101010 (6), and 111010/101000(5)

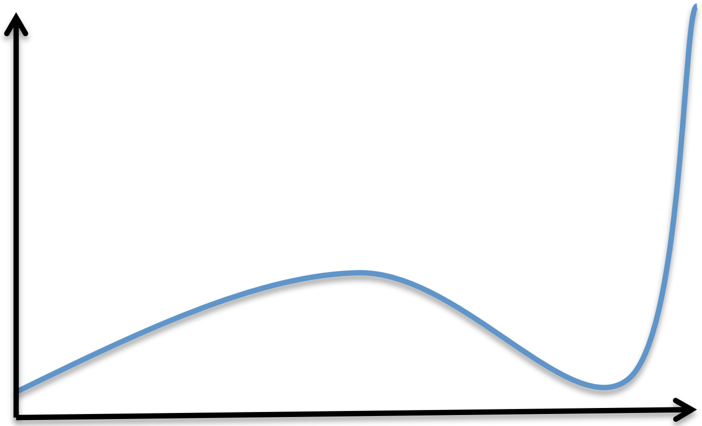
Finding the Global Maximum

- One simple idea: Hill-climbing with random restarts

- Keep running hill-climbing for a given number of times or we stop seeing improvements, starting at a random place each time

Still not quite what we want





Random restarts not likely to work well in some spaces (hilly or skewed spaces)

Bad Solutions

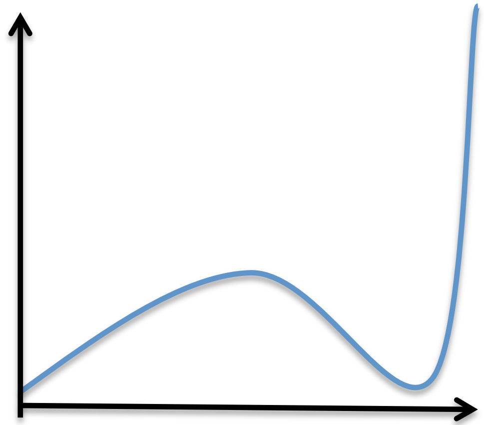
What if we could better check the entirety of the available space of representations?

- **Exhaustive Search**: Check every single point, grab the best (infinite/large H problem again)
- **Random Walk**: Keep making random moves till we stopping seeing anything better (sometimes better than exhaustive)

Simulated Annealing

Intuition: What if we sometimes took random steps at the start of search and took less and less as the search continued?

- We could better **explore** the space



Simulated Annealing

Think of it as a hybrid between random walk and hill-climbing.

Inspired by annealing in metallurgy/ blacksmithing

Simulated Annealing Pseudocode

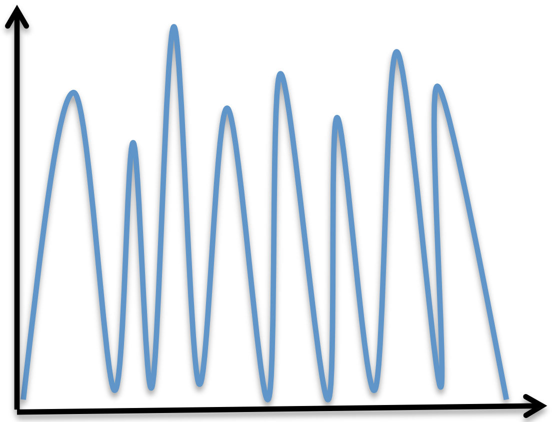
```
curr = random_selection_from_space()
score = heuristic(curr)
temperature = some_max_temperature
while temperature > 0 :
        next = random_neighbor(curr)
        temperature= schedule(temperature)
        changeH = heuristic(next)-score
        if (changeH> 0): curr = next, score = changeH+score
        else if (e^(changeH/temperature)>Rand()): curr = next, score = changeH+score
 return curr
```

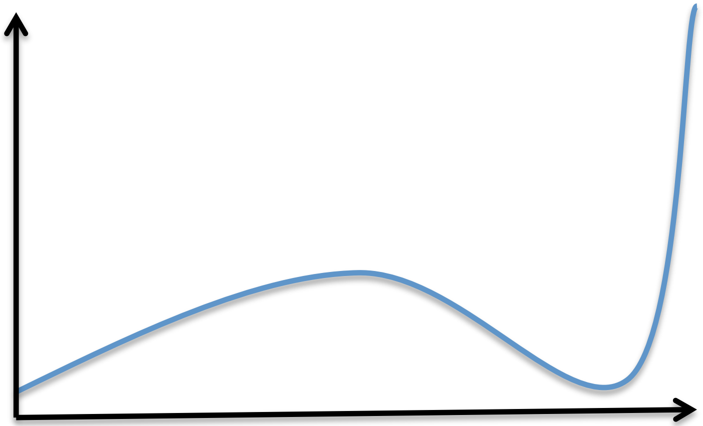
Simulated Annealing Behavior

- If temperature = ∞, behavior is random walk
- If temperature = 0, behavior is hill-climbing

- Key is to choose a good schedule to guide decrease of temperature.

Simulated Annealing Problem





Would Simulated Annealing do any better?

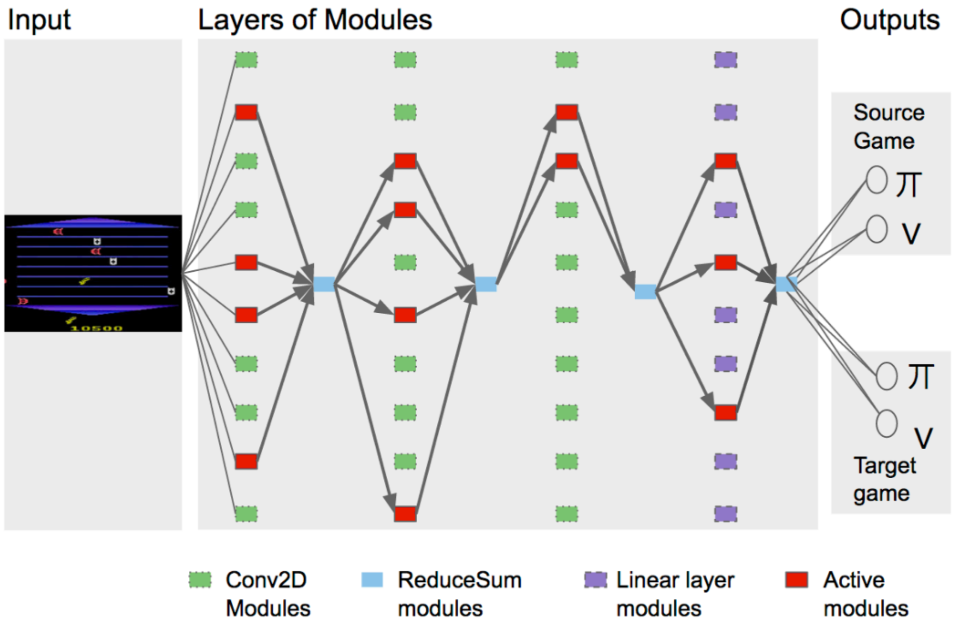
Answer: Sometimes to the right, not the left

Genetic Algorithms

Genetic Algorithms in the Wild

2006 NASA Spacecraft Antenna

Genetic Algorithms in the Wild

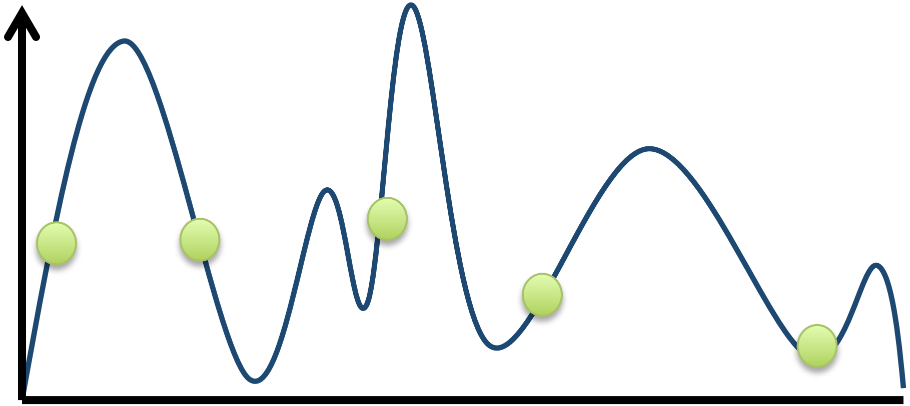


DeepMind: PathNet: Evolution Channels Gradient Descent in Super Neural Networks

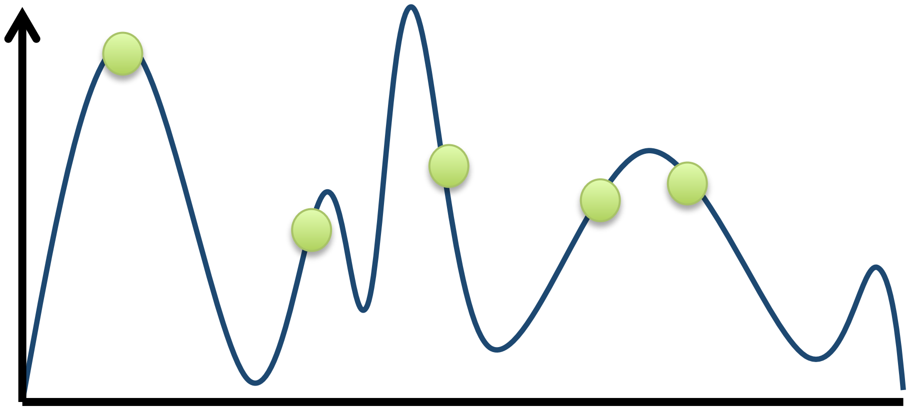
Genetic Algorithms

- Sometimes called evolutionary search
- Intuition: Instead of searching/walking from a single point -> many points spread across the search space
  - Use the knowledge gained from these points to take big, but likely better leaps

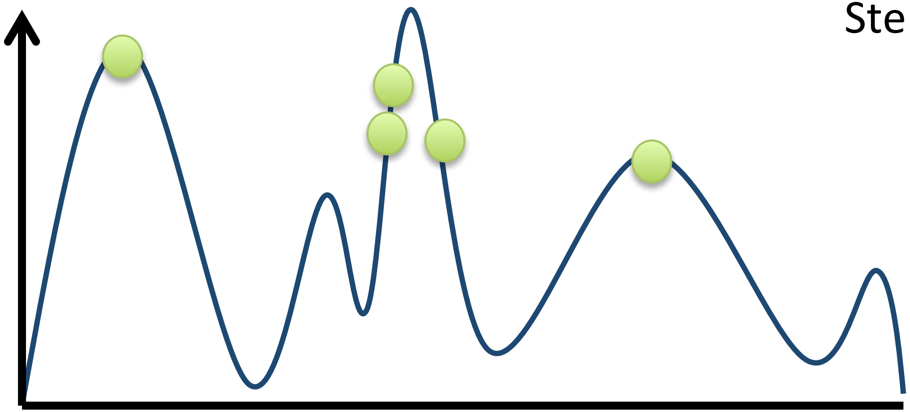
Genetic Algorithms Visualized



Initialization



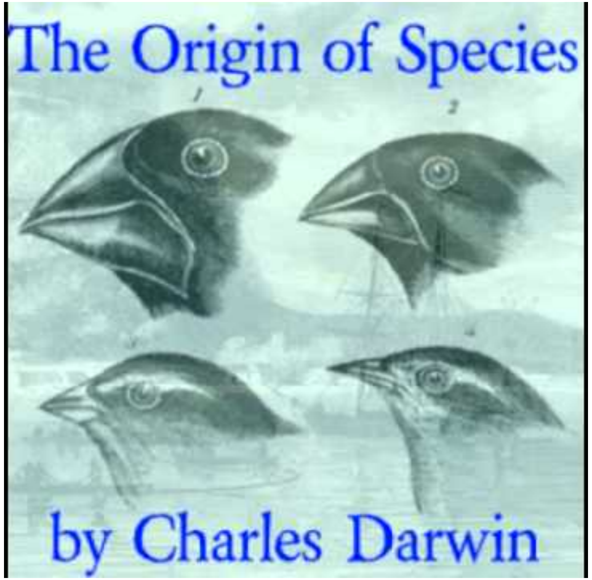
Step 2



Step 3

Genetic Algorithms: How the heck?

- Inspired by biological evolution
- Population of points
  - Undergoes mutation (random walk)
  - Reproduces (crossover)
  - Fittest survive (heuristic)



GA Pseudocode

population = set of random points of size *n*

time = 0

while **heuristic**(population)<*threshold* and time<max:

    time ++

    Mutate(population)

    population = **Crossover**(population)

    population = **Reduce**(population) *//back to size n*

return $\text{max}_{\text{heuristic}}$(population)

Key Functions

- **Mutate**: This is a random walk, according to some user-set probability replace a member of the population with a random neighbor
- **Crossover**: Mix two members of population selected according to their heuristic value
  - Population >= 2*n
- **Reduce:** Take the n best, or take the n new best

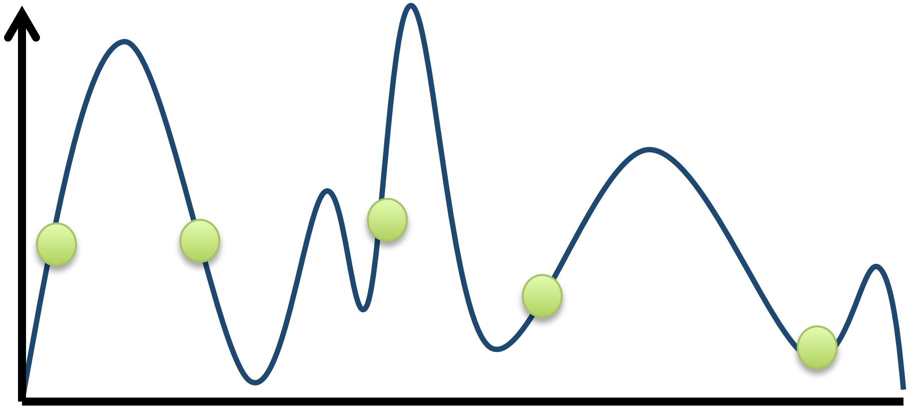
Crossover Strategies

Imagine we have the 6 bit strings from before. We could imagine many possible crossovers given two parents A and B.

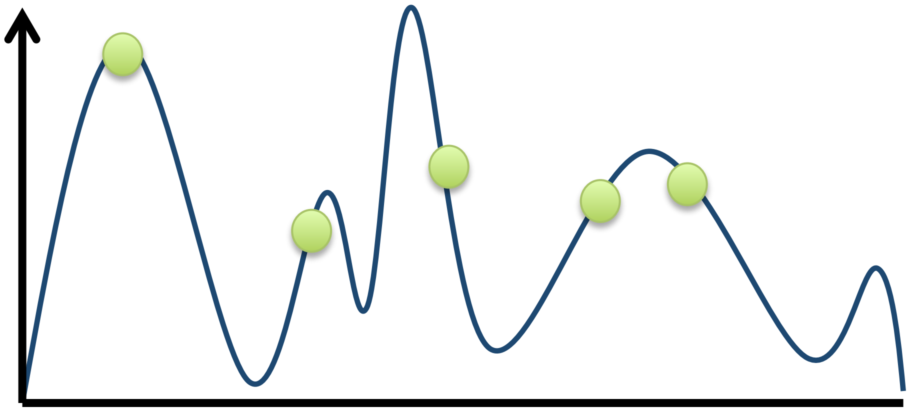
- <u>Single-point</u>: Pick some index i, take all elements of A before i, all elements of B >=I
  - 100100 and 111111 with i=4 -> 100111

Strategies continued…

- Two-point: pick two indexes i,j where j>i and j<length. Take all points [0,i) from A, all points [i,j] from B and all points [j+1, l) from A
  - 100100 and 111111 with i=2,j=4 -> 101110
- Uniform: For each index/variable, randomly select from both parents

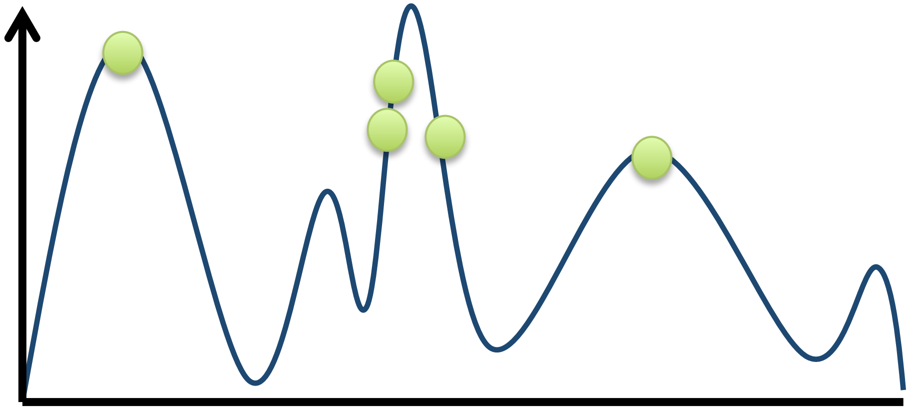


Initialization



Step 2

- What is n?
- What is the likely crossover function?
- What is the reduce function?

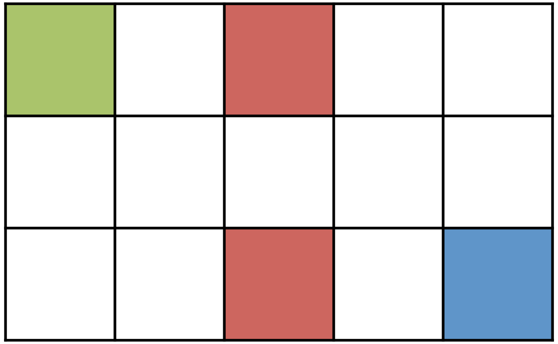


Step 3

GA Exercise

Problem: plans for a grid world. Start (green), get to blue w/ minimum cost. Red is lava (infinite cost).

Representation: sequences of length 10, actions down (1 cost), right (1), left (1), up (1), and wait (0)



Sequence: down, right, right, right, right, right, right, right, right, down

- Would get to goal with 10 cost.

Give me population size and heuristic, mutation, crossover and reduce functions.

GA Exercise (My Answers)

- Population size: 10 (arbitrary)
- Heuristic: Manhattan Distance to Goal – Cost
- Crossover: Single-point (first half of plan/ second half of another plan)
- Reduce: Take 10 best of parents and children (don't want to lose good plans)

Random Optimization Summary

- Hill-climbing: Strong simple approach, but weak to local maxima
- Simulated Annealing: With temperature we can get benefits of random walk and hill-climbing, but also some of their downsides
- Genetic Algorithms: With quite a bit of authoring we can get a robust solution to optimization problems
  - Called the 2nd best solution to anything